# **Makefile Complete Revision Guide**

## **1. Basic Syntax of a Makefile**

A Makefile consists of **targets, dependencies, and commands**:

# Basic structure:

target: dependencies

command

* **target**: The file to generate (e.g., an executable, object file).
* **dependencies**: Files required to build the target.
* **command**: Shell command to execute (must start with a TAB).

### **Example: Simple Compilation**

calculator: main.o math\_utils.o

g++ main.o math\_utils.o -o calculator

main.o: main.cpp math\_utils.h

g++ -c main.cpp -o main.o

math\_utils.o: math\_utils.cpp math\_utils.h

g++ -c math\_utils.cpp -o math\_utils.o

clean:

rm -f \*.o calculator

## **2. Variables in Makefile**

### **Defining and Using Variables**

CC = g++

CXXFLAGS = -Wall -Wextra -std=c++11

SRC = $(wildcard \*.cpp)

OBJ = $(SRC:.cpp=.o)

Use variables like:

$(CC) $(CXXFLAGS) -c $< -o $@

## **3. Pattern Rules (%)**

Pattern rules generalize the build process.

%.o: %.cpp

$(CC) $(CXXFLAGS) -c $< -o $@

* %.o → Any .o file
* %.cpp → Corresponding .cpp file
* $< → First dependency (source file)
* $@ → Target file (object file)

## **4. Automatic Variables**

| **Variable** | **Meaning** |
| --- | --- |
| $@ | Target name |
| $< | First dependency |
| $^ | All dependencies |
| $? | Dependencies newer than target |

Example:

calculator: $(OBJ)

$(CC) $(CXXFLAGS) $^ -o $@

Expands to:

g++ -Wall -Wextra -std=c++11 main.o math\_utils.o -o calculator

## **5. Wildcards (\*)**

Wildcards allow Make to automatically detect matching files.

SRC = $(wildcard \*.cpp)

OBJ = $(SRC:.cpp=.o)

* $(wildcard \*.cpp) finds all .cpp files in the directory.
* $(SRC:.cpp=.o) replaces .cpp with .o to create object file names.

This ensures that as new source files are added, they are automatically included in compilation.

## **6. Phony Targets (.PHONY)**

Used for targets that don’t generate files.

.PHONY: clean

clean:

rm -f \*.o calculator

## **7. Implicit Rules**

Make **automatically** knows how to compile .cpp → .o files without explicitly defining rules.

For example:

all: main.o utils.o

Make implicitly applies:

g++ -c main.cpp -o main.o

g++ -c utils.cpp -o utils.o

## **8. Advanced Implicit Rules**

Make can infer common operations based on file extensions. Instead of manually specifying:

main.o: main.cpp

g++ -c main.cpp -o main.o

We can **define an implicit rule** that applies to all .cpp files:

%.o: %.cpp

g++ -c $< -o $@

Now, Make will automatically apply this rule whenever an .o file is required but missing.

## **9. Handling Errors Gracefully**

all:

-mkdir bin # Ignore error if folder exists

g++ -c main.cpp -o main.o || echo "Compilation failed"

## **10. Debugging a Makefile**

Run in debug mode:

make VERBOSE=1

## **11. Complex Example (With Wildcards & Pattern Rules)**

CXX = g++

CXXFLAGS = -Wall -Wextra -std=c++11

SRC = $(wildcard \*.cpp)

OBJ = $(SRC:.cpp=.o)

TARGET = calculator

all: $(TARGET)

$(TARGET): $(OBJ)

$(CXX) $(CXXFLAGS) $^ -o $@

%.o: %.cpp

$(CXX) $(CXXFLAGS) -c $< -o $@

clean:

rm -f $(OBJ) $(TARGET)

### **Key Takeaways**

✅ Automate compilation with pattern rules. ✅ Use wildcards to detect files automatically. ✅ clean keeps the workspace organized. ✅ Use -k to keep compiling after errors.

This guide covers **everything** you need for your lab exam. Let me know if you need more practice questions! 🚀